A Category Theoretic Formalism for Abstract Interpretation

Prakash Panangaden
Prateek Mishra
Department of Computer Science
University of Utah
Salt Lake City, UT 84112

UUCS-84-005

Abstract: We present a formal theory of abstract interpretation based on a new category theoretic formalism. This formalism allows one to derive a collecting semantics which preserves continuity of lifted functions and for which the lifting functor is itself continuous. The theory of abstract interpretation is then presented as an approximation of this collecting semantics. The use of categories rather than complete partial orders eliminates the need for introducing two distinct partial orders and for introducing any closure operation on the allowable elements, as is necessary with powerdomains. Furthermore, our construction can be applied to any situation for which the underlying domains are complete partial orders, sice the domains are not further restricted in any way. This formalism can be applied to first order languages.

keywords and phrases: abstract interpretation, category theory, complete partial orders, denotational semantics, indeterminate functions, quasi-functor, strictness.

1. Introduction

Abstract interpretation is a general framework for justifying inference schemes aimed at the static determination of run time properties of programs. The original development of this framework was done by [Cousot 77]. Their formalism was developed in the context of imperative programs and was based on a lattice theoretic formalism. Recently, a number of attempts have been made to extend these ideas to applicative programs. In reasoning about applicative programs, two issues become important; first, non-termination needs to be addressed, which means that we need to work with domains rather than sets, second, the underlying domain may be non-flat if the language features *lazy* evaluation [Mycroft 80, Mycroft 83, Mishra 84]. Typically, the inference schemes used involve reasoning about the effect of functions on *arbitrary* sets of values rather than sets which are convex, this precludes the use of the Plotkin powerdomain construction. So far only limited results have been obtained; [Mishra 84] develops a theory that works when the simplified domains are finite, and [Mycroft 83] develops a formalism that works when the original domain is of finite height (i.e. all chains converge in a finite number of steps). The approach of the present paper is thus the most general available.

Our scheme uses a categorical formalism to obtain a theory

* that does not require that the sets be restricted to closed sets,
* for which the lifting process does not cause any continuity anomaly,
* which is not restricted to domains of finite height.

The power of the formalism derives from the use of morphisms in a category to express the approximation relation. Morphisms do not merely express the fact of approximation but give *precise details* of how two objects may approximate each other. This increased precision allows us to avoid the continuity problems of the other formalisms. Furthermore, because categories are equipped with a theory of limits, we are not

constrained to ensure that our models are equipped with a partial order. This means that we can essentially work with a preorder and still take limits; thus we are not forced to restrict our sets to be closed sets as is done in the standard powerdomain constructions [Plotkin 76], [Smythe 78].

Some of the standard terms that we use are defined in this paragraph. By *domain* we shall mean a complete partial order, by powerdomain we shall mean the constructions defined by [Plotkin 76] and others in which a complete partial order is constructed out of some of the subsets of a complete partial order. By the term *collecting semantics* we mean the extension of a semantics from functions defined on values to functions (actually functors, as we shall see) defined on *sets* of values. We shall use the symbol "?" to stand for the least element of a complete partial order; and "X" for the product domain constructor. By *abstract interpretation* we shall mean a "non-standard" semantics which is required to respect the standard semantics in an appropriate sense.

2. Previous Constructions

A general framework for the abstract interpretation of imperative programs was first developed in [Cousot 77]. Cousot&Cousot pointed out that many seemingly unrelated program analysis schemes can be formalized as semantic models that "approximate" (in a precise formal sense) the standard semantics of programs. The chief virtue of this approach is its emphasis on semantic soundness of program analysis schemes expressed in a single unified framework. Program analysis schemes are justified by viewing them to be abstractions of a canonical form of the standard semantics, the *collecting* or *static* semantics. In this setting the collecting semantics of imperative programs is the "lifting" of the standard semantics to the *powerset* of the underlying set of values.

The inadequacy of the powerset oriented approach, in modelling inference schemes for

applicative programs, was first shown by [Mycroft 81]. Inference schemes in the powerset oriented setting cannot capture termination and are therefore *weakly correct.* Functions in lazy applicative languages are non-strict and may carry out useful computation even when applied to a non-terminating expression. Hence inference schemes for applicative languages involve reasoning about termination (e.g. replacing call-by-name by call-by-value). Developing a theory of *strong* abstract interpretation forces the consideration of the underlying set of values to be a domain (set+order structure) rather than an unstructured set. Consequently, the appropriate power construction must capture *both* the underlying order structure of the domain as well as ordinary set theoretic inclusion. *Powerdomain* constructions familiar from work in the semantics of indeterminacy are therefore relevant.

In attempting to provide a general framework (the collecting semantics) for specifying such analysis, Mycroft proposed the use of the now classical *Plotkin* powerdomain as developed in [Plotkin 76]. The Plotkin powerdomain is restricted to modelling *bounded indeterminacy* [Apt 81] which implies that only selected subsets of the underlying domain are permissible and that in particular all infinite sets contain ?. This lack of expressive power sharply constrains the class of inference schemes that can be described in such a framework. For example, even simple inference schemes that involve describing infinite sets of values in the original domain by values in the abstract domain (e.g. analyzing integer valued applicative programs on the four point domain {POS, NEG, ZERO,?}) cannot be expressed in this framework. Hence the Plotkin powerdomain is *not* suitable for specifying the collecting semantics.

In [Mishra 84] several practical examples of static inference involving streams (a non-flat domain) were developed in the framework of abstract interpretation. It was pointed out that a collecting semantics for inference over non-flat domains posed difficult

problems, as a well-behaved powerdomain construction over a non-flat domain with a sufficiently rich collection of sets was not available.

In [Mycroft 83] the collecting semantics for a restricted class of domains (domains with finite ascending chains - *fdcpos*) was developed. The construction remedied the problems with the inadequate expressive power of the Plotkin construction as described above. However, a number of problems remained. The continuity of functions "lifted" to the powerdomain was lost; hence the framework forced the consideration of *monotonic* functions. The "lifting" functor from the original function space to the function space over the powerdomains was also not continuous. As the collecting semantics does not involve any *loss* of information as compared to the standard semantics, the loss of continuity appears counter-intuitive. Further it was not clear how the construction might be adapted to describing inference schemes over domains which did not possess the fdcpo property (e.g. streams).

In [Abramsky 83] a category-theoretic formulation using multidomains (multi-set domains) was proposed as a suitable model for (unbounded) indeterminate computation on arbitrary domains. From the point of view of static inference this approach is unsatisfactory, since multi-sets over even finite domains do not possess the finite chain property[1]; hence static inference schemes are not effective.

3. The Powergraph Construction for Determinate Functions

In this section we introduce a preliminary construction that provides a suitable collecting semantics for a first order language that does not permit the use of indeterminate functions. We shall see that this construction does achieve many of the

---

[1] all chains converge in a finite number of steps

formal requirements for a collecting semantics. However, the inability to handle indeterminate functions will render it useless for our purposes. This construction does play a useful pedagogical role in indicating the generalizations needed, and the complete construction, given in the next section, will lean heavily on the material of this section. This construction does not use categories and will serve to define the limits of what one can do in the context of complete partial orders.

The first key observation we make is the following; when a function is extended pointwise to act on sets of values, the precise correspondence between argument and result is preserved only by the singleton sets. Thus, with larger sets, one can no longer determine exactly which pairs of values correspond. This is the major reason for the breakdown of continuity that was discussed in the last section. This observation suggests that instead of lifting the action of functions from individual values to sets of values, we should instead lift the *graph* of the function to the collection of all its possible subgraphs. Accordingly this construction is called the *powergraph* construction.

Given domains D and E, we may reinterpret functions f:D--->E as taking an element d of D to the pair <d,f(d)>, in other words a point does not get mapped to its image but to the corresponding point in the graph of the function. We shall denote these reinterpreted functions f^g. These functions are defined as acting from the domain D to D X E. To make the formalism more symmetric we shall take instead of D the domain D X ?, where ? is the one-point domain. It is easy to see that D X ? is canonically isomorphic to the domain D. The arity of the reinterpreted functions is:

D X ? ---> D X E.

It is easy to see that f^g is monotonic and continuous iff f is monotonic and continuous.

We now construct the powergraph domains as consisting of suitable subsets of the cross product domains defined above. As a first attempt, we begin with the observation

that we would like to reason about the action of functions on arbitrary subsets of the domain D, so the "power" construction should include all subsets of the domains D X ? and D X E. The partial order should try to reflect the following intuition: a subgraph may "improve" by expanding in size or by giving more precise information about the result values. This suggests the following partial order on D X E:

Def. 3.1 $K_1 <= K_2$ if there exists a function ι from K_1 to K_2 such that if $\iota(x) = y$ then $\pi_1(x) = \pi_1(y)$ and $\pi_2(x) <=_E (y)$.

Unfortunately, this is not a partial order on the collection of all subsets of D X E. Consider the following two subsets of D X E:

$K_1 = \{<d,e_1>,<d,e_3>,<d,e_5>,....\}$
$K_2 = \{<d,e_2>,<d,e_4>,<d,e_6>,....\}$
where d is some element of D and $\{e_1,e_2,e_3,...\}$ forms an increasing chain in E. Then clearly we can find maps from K_1 to K_2 and vice versa satisfying the above conditions.

A further objection to including all subsets of D X E is that several of these subsets cannot possibly be the subgraphs of functions. Thus we are led to the following restriction on the subsets of D X E to be included in our power construction: all the sets must have at most one occurrence of an element from D as the first component of a pair. The putative partial order defined above is now really a partial order. We shall write **G**(D,E) to denote this domain. Formally we define:

Def 3.2 **G**(D,E) is the collection of all subsets of D X E satisfying if $K \in$ **G**(D,E) and $x,y \in K$ then $\pi_1(x) /= \pi_1(y)$.
G(D,E) is equipped with the partial order defined above.

Theorem 3.3: **G**(D,E) is a complete partial order.

Proof: **G**(D,E) is clearly a poset. The empty set is the least element. Consider a directed set of elements of **G**(D,E), $\{K_a\}$ where the αs belong to some indexing set. For each pair K_a,K_b with $K_a <= K_b$ there is an injection ι_{ab} from K_a to K_b satisfying the conditions above. Furthermore, this injection is the only one between K_a and K_b satisfying the conditions of Def 3.1. It then follows that if $K_a <= K_b$ and $K_b <= K_c$ then $\iota_{ac} = \iota_{ab}$ o ι_{bc}. Now we construct the least upper bound for the directed set. Consider any element d of D which

appears as the first component of a pair in any of the sets K_a. The element d will appear at most once in any pair in any single set. Take all pairs which have d as the first component. The maps ι_{ab} will connect all these pairs into a directed subset of D X E. Take the limit of this directed set in D X E, we know this limit exists because D X E is a complete partial order. We obtain a pair <d,e>, the set K is obtained by collecting all such pairs for every element of D that occurs as the first component of a pair in any of the K_a. It is easy to see that the K so constructed is the least upper bound of the directed set $\{K_a\}$.

The limit constructed above is in fact a *direct limit* construction in the category theoretic sense. When we generalize **G**(D,E) to a category we will essentially use the same construction to obtain limits. The complete partial order **G(D,?)** is easily seen to be canonically isomorphic to the domain of *all* subsets of D ordered by inclusion. Thus our powergraph construction allows us to reason about arbitrary subsets of D. We now show that the "lifting" of functions from D to E has many pleasing properties.

Let f^g be a function from D X ? to D X E. This function can be extended pointwise to obtain a function F^g from **G**(D,?) to **G**(D,E). It is clear that F^g is continuous and monotonic iff f^g is also continuous and monotonic. It is also true that the lifting process is itself monotonic and continuous. The fact that it is monotonic is easy to see so we shall only prove continuity explicitly.

Theorem 3.4: Let f^g_n be a chain of functions that converges to f^g. Let S be any subset of D X ?. Then the sequence $F^g_n(S)$ converges to $F^g(S)$ in **G**(D,E).

Proof: Consider any pair <d,?> in S. In every set in $F^g_n(S)$ there is a pair of the form $<d,f_n(d)>$ and this will in fact be the only pair which has d as its first component in each set. Furthermore the map ι_n from $F^g_n(S)$ to $F^g_{n+1}(S)$ will map $<d,f_n(d)>$ to $<d,f_{n+1}(d)>$. The sequence $\{<d,f_n(d)>\}$ thus forms a chain in D X E connected by the ι_n maps. The

least upper bound of this chain is clearly $<d,f(d)>$ or $f^g(<d,?>)$. Thus the least upper bound of $F^g_n(S)$ will consist of exatly those elements of the form $<d,f(d)>$ for each $<d,?>$ in S. This set is exactly the set $F^g(S)$. We should have actually shown this for a directed set of functions rather than a chain but the proof goes through in exactly the same way. Thus the lifting action is continuous.

We now consider the problem of embedding the domains D X ? and D X E into the domains G(D,1) and D,E. We would like these embedding maps to be continuous and monotonic. Since G(D,1) is canonically isomorphic to the domain of subsets of D ordered by inclusion we can define an embedding function:

Φ:[D X ?] ---> [D X E] given by $\Phi(d) = \{<d',?> | d' <=_D d\}$.

This function is clearly continuous and monotonic. If we attempt to use a similar definition for embedding D X E into G(D,E) we will not succeed because the embedding of $<d,e>$ should be the cartesian products of the sets $\{d' | d' <= d\}$ and $\{e' | e' <= e\}$, this set cannot be a member of G(D,E) since it will, in general, contain several pairs with the same first component. Thus we cannot embed the domains D X ? and D X E into the corresponding powergraph domains. In the next section, we will be able to perform the embedding by using the categorical construction to allow more sets in G(D,E).

An important point that we now address is the issue of function composition. Suppose that we are considering functions from D to D instead of from D to E. Then we can compose functions, this composition will have to be carried over into out graph domains. Let f,h be functions from D to D. Then f^g and h^g will be functions from D X ? to D X D. The composition of f^g and h^g, written $f^g * h^g$, must satisfy:

$(f \circ h)^g(<d,?>) = <d,h(f(d))> = (f^g * h^g)(<d,?>)$.

Expressed in terms of projections and pairings we may write:

$(f^g * h^g)(<d,?>) = <d,\pi_2(h^g(<\pi_2(f^g(<d,?>)),?>))>$

$=<\pi_1(<d,?>),\pi_2(h^g(<\pi_2(f^g(<d,?>)),?>))>$

To express this as ordinary function composition we introduce the following two combinators:

$P_0(f^g) = \lambda x.<\pi_2(f^g(x)),?>$
$P_1(f^g) = \lambda x.<\pi_1(x),\pi_2(f^g(x))>$

We can now express the * composition operator as:

$f^g * h^g = P_1 \circ h^g \circ P_0 \circ f^g$.

Once we are equipped with function composition, we can express the meaning of recursively defined functions via standard fixed point theory.

The collecting semantics for a first order language can now be obtained in a straightforward fashion. Given the standard semantics as a function μ from **Syn**, the set of syntactic forms, to the semantic domains we then interpret functions as the pointwise extensions of their graph analogues and arguments to functions as arbitrary sets of values from the relevant domain. The collecting semantics thus obtained involves no loss of information with respect to the standard semantics and preserves continuous and monotonicity.

To illustrate the absence of the continuity anomaly noted by [Mycroft 83] we consider their example, discussed in section 2. Let f_i denote the sequence of functions:
$f_i(n)$ = if n <= i then 1 else ?.

Now consider the action of this sequence of functions on the set N. First we express each function in its graph form and then lift these to obtain the sequence of functions F^g_i from $\mathbf{G}(N_?,?)$ to $\mathbf{G}(N_?,N_?)$. When we apply the functions F^g_i to N we obtain a sequence of members of $\mathbf{G}(N_?,N_?)$ which form a chain. If we consider any integer *m* we see that the sequence of injections that establish the approximation relation in $\mathbf{G}(N_?,N_?)$ forms an unique "thread" through *m* with the second component of each pair in such a thread

being ? until the mth element is reached, whereupon the second components all become 1. Taking the least upper bound in $\mathbf{G}(N_?,N_?)$, we get the set N X {1}. This is exactly what we would get if we took the least upper bound of the sequence of functions f^g_i and lifted the result and then applied the lift of the limit function to N (actually to N X {?}).

4. The Categorical Construction

In this section we will generalize the construction described in the prior section so that we can handle indeterminate functions and provide embeddings of the underlying domains in the "power-domains". The new ingredients needed are the introduction of categories rather than domains and a relaxation of the restrictions on the sets in **G**(D,E). The use of categories in semantics has been advocated by [Lehmann 76] and by [Abramsky 83]. Our approach uses a different construction for the objects and morphisms and also defines limits in a different fashion. The intuition behind the use of categories is that objects may approximate other objects in different ways, the morphisms express these different senses of approximation. The purely order theoretic approach merely expresses the fact of approximation.

We define the *category* of subgraphs of a pair of domains D, E by:

Def 4.1 PG(D,E), the category of subgraphs of the pair of complete partial orders D and E has objects consisting of all subsets of D X E and morphisms are functions which satisfy the following condition: If f is a morphism from P to Q then if f(x)=y then $\pi_1(x) <= \pi_1(y)$ and $\pi_2(x) <=_E \pi_2(y)$.

Unlike the preceding construction, we do not require that the objects contain at most one element with a given first component, thus we allow the objects to be subgraphs of *relations*. A consequence of this is that the morphisms are not injections. It is clear that morphisms may exist in both directions between a pair of objects. Thus the morphisms do not define a partial order. The category **PG**(D,?) has exactly the same definition as **G**(D,E), with E replaced by the one point domain ?. The category **D,?** no longer is the

powerset of D ordered by inclusion.

Now we interpret functions from D to E as *functors* between **PG**(D,?) and **PG**(D,E). Let f be a function from D to E. The functor F = **Lift**(f) is defined in the following way:

Def 4.2: Let P be an object of **PG**(D,?). F(P) is the object $\{<d,e> | <d,?> \in P \text{ and } e = f(d)\}$ of **PG**(D,E). Let κ be a morphism from P to Q in **PG**(D,?). The morphism F(κ) from F(P) to F(Q) maps each pair $<d,f(d)>$ in F(P) to the pair $<\kappa(d),f(\kappa(d))>$ in F(Q).

The fact that functions from D to E can be interpreted as functors is the categorical way of expressing monotonicity. Notice that F(κ) would not be a morphism if f were not monotonic.

We now prove that the lifting of a continuous function f, yields a continuous functor F. Let X_a be a family of objects in **PG**(D,?) with morphisms μ_{ab} from X_a to X_b forming a commuting diagram and having X as the direct limit with morphisms ν_a from X_a to X. Apply F to this commuting diagram to obtain a corresponding commuting diagram in **PG**(D,E), the claim is that F(X) is the limit of the diagram in **PG**(D,E). Consider any element $<d_a,e_a>$ in $F(X_a)$, this element will be part of a chain in D X E formed by the morphisms $F(\mu_{ab})$ in **PG**(D,E). Since D X E is a complete partial order we know that this chain will have a limit $<d,e>$; thus $<d,e>$ is a member of the limit object in **PG**(D,E) and all members of the limit object in **PG**(D,E) arise in this fashion. Furthermore, we know that e = f(d) where d is the limit of the chain formed by the d_as. Thus we know that $<d,?>$ must be in the object X, and thus $<d,e>$ must be in F(X). It is easy to see that the morphisms behave in the required way under F. Thus F is continuous.

This reincarnation of ordinary functions as functors does not carry over to the situation where the functions are indeterminate. Suppose X and Y are objects in **PG**(D,?) with κ a morphism from X to Y. Suppose r is a relation from D to E. Define a map R from the objects of **PG**(D,?) to the objects of **PG**(D,E) in the following way. The action of R is to

form the collection of all pairs which are related with first components appearing as the first component of a pair in the original object. Thus

$R(X) = \{<d,e> | <d,?> \in X \text{ and } <d,e> \in r\}$.

Thus relations can be interpreted as *maps* between the set of objects in **PG**(D,?) and the set of objects in **PG**(D,E); the fact that it is not a functor can be understood in the following way. A morphism expresses approximation between objects in a *detailed* way, thus when one is dealing with a function applied to sets of values it is natural to expect that one can establish a precise correspondence between the original objects and the image objects. With relations, however, a given value may be related to several different values, and therefore a natural morphism cannot be found. What computability concept should be applied to relations ? Our proposal is the following:

Def 4.3: A relation r is called *quasi-monotonic* if whenever $d <= d'$ then for every x in r(d) there exists a y in r(d') with $x <= y$.

The concept analogous to continuity that we propose is:

Def 4.4: A relation r is called *quasi-continuous* if given a directed set $\{d_a\}$ (the *a*s belong to some index set) which converges to d then every directed set $\{e_a\}$ that can be formed by choosing appropriate elements from the sets $r(d_a)$ converges to an element belonging to r(d).

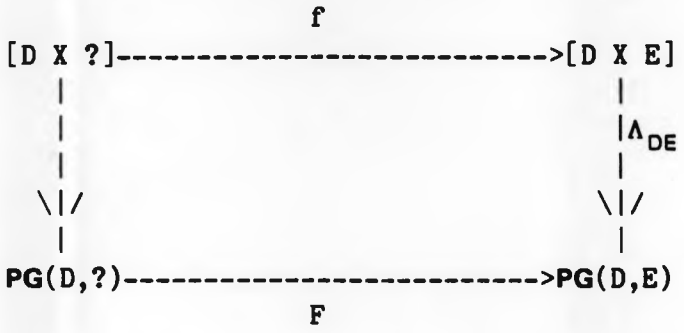
Using these concepts, what can be said about the maps R that arise as the lifting of quasi-monotonic and quasi-continuous relations ? Given objects X and Y in **PG**(D,?) with a morphism κ in **PG**(D,?) from X to Y we obtain objects R(X) and R(Y) in **PG**(D,E). There is no canonical morphism in **PG**(D,E) from R(X) to R(Y), but if r is quasi-monotonic then we can guarantee that there exists *some* morphism from R(X) to R(Y). This is easy to see, for every $<d,e>$ in R(X) there must be a $<d',e'>$ in R(Y) with $d <= d'$ and $e <= e'$ by the definition of quasi-monotonic. Similarly, the quasi-continuous condition means that if we apply R to every member of a diagram in **PG**(D,?) with a direct limit X, we can obtain, in general, several different diagrams in **PG**(D,E) with direct limits. The limit objects will all be contained in R(X). An R with such properties will be called a *quasi-functor*.

The category **PG**(D,E) has ϕ as the initial object. Limits in this category are constructed in essentially the same way as in the preceding section. We illustrate the new features that arise by allowing subgraphs of relations via a few simple examples. Let $\{e_n\}$ be a chain in E and d a fixed element of D. Consider the sequence of objects $\{S_n\}$ in **PG**(D,E), where $S_n = \{<d,e_m> | m=>n\}$. Define morphisms ι_n from S_n to S_{n+1} by $\iota_n(<d,e_m>)=<d,e_{m+1}>$. The limit of this system is $\{<d,e>\}$, where e is the least upper bound of $\{e_n\}$. Now suppose we define the following family of sets. The set S_n contains all pairs of the form $<d,e_m>$ for odd m if n is odd and even m if n is even. We can use the same maps as we used before to obtain a direct system. The direct limit of this system is also $\{<d,e>\}$.

We now need to argue that the lifting process is itself continuous. Suppose that f_i is a sequence of functions from D to E which converges to f. Suppose that the corresponding functors are F_i from **PG**(D,?) to **PG**(D,E). Let X be an object of **PG**(D,?). We need to show that the "sequence" of objects $F_i(X)$ converges to F(X). An immediate problem arises; in categories limits are defined for *diagrams* not for sequences. It makes no sense, a priori, to talk about the limits of sequences of objects alone; we need to have a *diagram*. We claim that it is natural to construct the following diagram. Define functions ϕ_i from X to each of the $F_i(X)$s by $\phi_i(<d,?>) = <d,f_i(d)>$. These functions will all be bijections. Now we define morphisms μ_{ij} from $F_i(X)$ to $F_j(X)$ in **PG**(D,E). If we require that the entire diagram commute it is easy to see that the only possible morphisms are those that agree on the first components. The situation now reduces to that of the previous section, where we saw that the limit in **PG**(D,E) was indeeed F(X) where F is the functor corresponding to X. Thus limit is called the X-indexed limit. We claim that this is the natural extension of the limit concept for sequences of continuous functions from one complete partial order to another.

The domains D X ? and D X E can be embedded into the categories **PG**(D,?) and **PG**(D,E) in the following way. We define a map Λ_D from [D X ?] to **PG**(D,?) by $\langle d,? \rangle \mapsto \{\langle d,? \rangle\}$ and similarly we can define Λ_{DE} from [D X E] to **PG**(D,E). It is easy to see that these maps are continuous and monotonic.

We thus obtain the following diagram relating the various categories and domains.



The functions f are lifted as before to functors from **PG**(D,?) to **PG**(D,E). The diagram commutes so we see that the collecting semantics is precisely equivalent to the standard semantics.

Since we are not interested in modelling indeterminate operators in the standard semantics we do not need to discuss the union operation. It is worth noting that the union operation can be expressed as ordinary set union in our categories. This is because our use of categories has allowed us to handle arbitrary sets; we do not require any particular closure property to be obeyed. This offers the prospect of being able to develop a general theory of indeterminate operators along the categorical formalism of the present paper. Such an investigation has been initiated and results will be reported in a forthcoming report.

5. Abstract Interpretation

In this section we show that abstract interpretation can be formally related to the collecting semantics as described in the last section. This will provide a general framework for showing that particular abstractions are in fact semantically sound. The key idea of abstract interpretation is that the abstracted domains should be simplified in some suitable sense. Crudely speaking, this means that the abstracted domains should have fewer elements than the original domains. Thus an element of the abstracted domain should represent a set of elements from the original domain. In this section we shall use D and E to stand for the "standard" semantic domains and A and B for the corresponding simplified domains.

An *abstract interpretation* is formally defined through a pair of quasi-functors from **PG**(D,?) to **PG**(A,?) and from **PG**(D,E) to **PG**(A,B); the quasi-functors are usually written as γ_1 and γ_2 respectively. To see why the maps cannot be functors consider the following situation. Suppose that the object X in **PG**(D,?) consists of the pairs $<d_1,?>$ and $<d_2,?>$ and suppose that the object Y in **PG**(D,?) consists of the pairs $<d_3,?>$ and $<d_4,?>$. Suppose further that there is a morphism σ from X to Y with $\sigma(<d_1,?>) = <d_3,?>$ and $\sigma(<d_2,?>) = <d_4,?>$. Now, under an abstraction it is possible that d_1 and d_2 get identified but that d_3 and d_4 do not. While it is not clear <u>what</u> morphism should be established from the image of X to the image of Y, it is clear that <u>some</u> morphism can be found. Thus, by using a quasi-functor rather than a functor, we are acknowledging that certain details of how a computation would progress are lost but we respect as much of the structure of the original category as we can.

The abstraction maps, γ_1 and γ_2, must be restricted further; they must commute with the set union operation in **PG**(D,?) and **PG**(D,E). This means that the actions of the abstraction maps are given by their actions on the singleton sets and are essentially

pointwise extensions of abstraction functions α_1:D--->A and α_2:B--->E. The most important condition that we need to express is the *acceptability* condition of [Mishra 84]. This condition arises from the need to control the behavior of the *inverse* of the abstraction maps. We define maps β_1 and β_2 from **PG**(A,?) to **PG**(D,?) and from **PG**(A,B) to **PG**(D,E) respectively. β_1 is defined as:

Let U be an object of **PG**(A,?), then
$\beta_1(U) = \{<d,?>|\gamma_1(\{<d,?>\}) \quad U\}$.

The map β_2 is defined analogously. These maps are called the *concretization* maps. We note that these maps, β_1 and β_2, are *not* quasi-functors; essentially this is because the pre-images of sets in **PG**(A,?) may contain limit elements. We will therefore not be able to find morphisms between the preimages of related sets, in general. This arises because our abstraction may have an *infinite* element being represented by a finite element of the abstracted domains. We introduce an operator **fin** which takes an object and returns the subset consisting of all the *finite* elements. The maps β_1 and β_2 are required to satisfy:

Def 5.1:if U,V are objects of **PG**(A,?) (resp. **PG**(A,B)) and there exists a morphism from U to V then there exists a morphism from **fin**(β_1 (resp. β_2)(U)) to **fin**(β_1 (resp.β_2)(V)).

Thus if the domains A and B were to contain only finite elements, then the maps β_1 and β_2 would indeed be quasi-functors.

Given a function from D X ? to D X E which lifts to the functor F from **PG**(D,?) to **PG**(D,E), we must show how to interpret this as a quasi-functor from **PG**(A,?) to **PG**(A,B). First we note that β_1 and β_2 define *closure* operators on D X ? and D X E. This may be seen in the following way. Since γ_1 is quasi-continuous we see that if all the members of a directed set in D X ? are abstracted to a particular element of A X ?, say z, then the least upper bound of that directed set must also be abstracted to z. Now consider the action of β_1 o γ_1 on objects in **PG**(D,?). This map satisfies the conditions for a *closure operator* [Cohn 65]. The closed sets in **PG**(D,?) are closed under least upper bounds in D

X ?. Similarly, the composition β_2 o γ_2 defines a closure operator on the objects in **PG**(D,E). Now given a functor F from **PG**(D,?) to **PG**(D,E), we define the associated quasi-functor R_F from **PG**(A,?) to **PG**(A,B) by:

Def 5.2:$R_F = \beta_1$ o F o γ_2.

We need to demonstrate that this is indeed a quasi-functor. We shall explicitly show that R_F is quasi-monotonic, showing quasi-continuity is essentially similar. Let U and V be objects in **PG**(A,?) with a morphism κ from U to V. Now the objects $\beta_1(U)$ and $\beta_1(V)$ in **PG**(D,?) will not, in general, have a morphism between them, but **fin**($\beta_1(U)$) and **fin**($\beta_1(V)$) will have some morphism between them. Furthermore, the sets $\beta_1(U)$ and $\beta_1(V)$ are closed; under the action of F they stay closed, because F is a functor. Once again, there will not be a morphism from $F(\beta_1(U))$ to $F(\beta_1(V))$, but there will be a morphism from **fin**($F(\beta_1(U))$) to **fin**($F(\beta_1(V))$) in **PG**(D,E). Because γ_2 is a quasi-functor there will be a morphism from γ_2(**fin**($F(\beta_1(U))$)) to γ_2(**fin**($F(\beta_1(V))$)) in **PG**(A,B). Since the closures of **fin**($F(\beta_1(U))$) and **fin**($F(\beta_1(V))$) generated by β_2 o γ_2 are precisely $F(\beta_1(U))$ and $F(\beta_1(V))$ respectively, we see that $\gamma_2(F(\beta_1(U))) = \gamma_2(F($**fin**$(\beta_1(U))))$, the same reasoning can be applied to V. Now we already know that we can find a morphism from $\gamma_2(F($**fin**$(\beta_1(U))))$ to $\gamma_2(F($**fin**$(\beta_1(V))))$, it follows then that we can find a morphism from $R_F(U)$ to $R_F(V)$ in **PG**(A,B).

Because the composition of a quasi-functor with a quasi-functor is a quasi-functor we can reinterpret functors from **PG**(D,?) to **PG**(D,E) as quasi-functors from **PG**(A,?) to **PG**(A,B) and compose them freely. Furthermore, since direct limits exist in the categories **PG**(A,?) and **PG**(A,B) we can take limits, and by the definition of R_F, the limits we get are a superset of the limits that we would obtain by taking limits in **PG**(D,E) and then projecting to **PG**(A,B) using γ_2.

Suppose we have a language L, with certain primitive functions P_i and a standard semantics expressing the P_is as continuous functions from D to E. We can then obtain the

collecting semantics as we have described in this section by going to the categories **PG(D,?)** and **PG(D,E)** and expressing functions as functors. The abstract interpretation is obtained by finding appropriate simplified domains A and B and then constructing **PG(A,?)** and **PG(A,B)** and finding maps γ_1, γ_2, β_1 and β_2 which satisfy the conditions described in this section. The interpretation of the primitive functions is obtained by forming the R_{P_i}s as described above, and then using composition of quasi-functors to obtain more complex functions. The semantic soundness of the abstract interpretation is thus tightly coupled to the standard interpretation given by the collecting semantics.

6. Applications

7. Applications

We briefly describe two static inference schemes that have appeared in the literature in our framework. In [Mycroft 81] Mycroft developed conditions under which function evaluation could be safely performed using call-by-value in place of call-by-need; we will call this analysis *strictness* inference. In [Mishra 84] Mishra developed a framework for *fine grain* type checking of functions defined on stream domains, called *minor signature* inference.

7.1. Strictness inference

Consider an applicative language defined on integers and booleans (both flat domains). Given function *f*, a sufficient condition for replacing call-by-need by call-by-value in evaluating an application of *f* is:

f:? = ?

As explained above we abstract the collecting semantics F:PG(N,?) -> PG(N,N) to a simplified domain. As we are interested in the action of functions purely in terms of defined and undefined values, we consider the two point domain TWO = {UNDEF,DEF}

with UNDEF <= DEF. The abstraction maps are:

α_1:S = { <UNDEF,UNDEF> | S = { <?,?> } }
α_1:S = { <DEF,UNDEF> | <d,?>ϵS and d $\neq$? }
α_1:S = { <DEF,UNDEF> , <UNDEF, UNDEF> |
<?,?>ϵS and <d,?>ϵS d $\neq$? }

α_1 (α_2 similarly) maps all defined integers to DEF; the undefined element is mapped to UNDEF. Both the abstraction and concretization maps are well behaved and therefore induce a mapping from PG(N,?) -> PG(N,N) to PG(TWO,?) -> PG(TWO,TWO).

For any function definition f = E[f] defined on the integers, we write f = R_E[f] as the corresponding function definition over the abstracted domains. In going from *E* to R_E we are simply reinterpreting all primitive functions over the new domain (i.e. primitive function *c* is interpreted as R_c). The test for strictness (for description of the implementation and other details see [Mycroft 81]) is then simply:

? ϵ $\lim_i$ R_F^i[?]:{ <UNDEF,UNDEF>}

7.2. Minor signature inference

Consider an applicative language defined on streams; in particular we assume a lazy interpretation of the *integer* type defined below as our example.

type integer = zero + succ[integer]

Note that under the lazy interpretation expressions of the form x = succ[x] have a solution: the infinite list of successors.

Our interest lies in determining whether functions defined on the integer type are defined for every term described by the above type-equation. If they are not defined (e.g. the *subtract1* function) we require a characterization of the terms for which the function yields an error (e.g. zero for *subtract1*). The particular abstract domain we use is drawn

from [Mishra 84] consists of the *constructors* (written *Con*) used in a particular expression. Figure I below displays the structure of the domain:

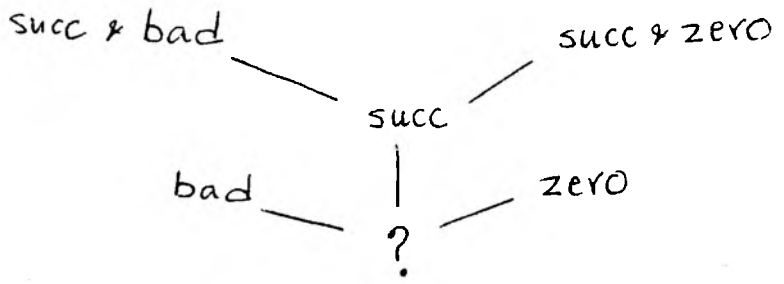


Figure I

The abstraction map from PG(integer,?) to PG(Con, ?) is defined by:

α_1:S = E
where
?∈E if ?∈S.
succ & zero∈E if a term composed of *succ* and *zero*∈S.
succ∈E if a term composed of *succ*∈S.
zero∈E if zero∈S.
bad∈E if bad∈S
succ & bad∈E if a term composed of *succ* and *bad*∈S.

α_1 (α_2 similarly) maps an integer into the set of constructors used to construct it. Bothe abstraction and concretization maps are well behaved and induce a mapping from PG(integer,?) -> PG(integer,integer) to PG(Con,?) -> PG(Con, Con).

Computation of the minor signature follows in the same manner as strictness inference. Algorithmic details can be found in [Mishra 84]. Below we display the function *subtract1* and its *minor signature* on the domain *Con*:

```
subtract1:succ[x] = x

subtract1:{<zero,?>} -> { <bad,zero> },
          {<succ,?>} -> { <succ,succ>},
          {<succ & zero,?>} -> {<succ & zero,succ & zero>,
               <zero,succ & zero>}
          {<succ & bad,?>} -> {<succ & bad, succ & bad>,
            <succ & bad,bad>},
          {<bad,?>} -> {<bad,bad>}
```

The minor signature for *subtract1* clearly indicates that the function is not defined for all terms belonging to *integer*; in particular it yields an error when applied to the term *zero*.

Finally, we note that as our approach does not require the identification of sets with their *convex* closures, the precision of the inference in the current framework is an improvement over that described in [Mishra 84].

8. Conclusions

The formalism of the present paper represents a departure from the standard approach of modelling computational progress via a complete partial order. [Abramsky 83] has also introduced a categorical approach, but he is forced to introduce *multi-sets*; these are unsuitable for abstract interpretation because even if the abstracted domain is finite the multi-set category will be infinite and thus useless for compile time analyses. We have also expressed computability concepts for indeterminate operators via the quasi-functor concept. In [Panangaden 84] we have applied the formalism of the present paper to discuss a semantic theory of indeterminate operators.

We believe that the formalism of the present paper will be useful in a number of contexts. In particular we are exploring the semantics of *set abstraction*, the semantics of dataflow nets that have indeterminate operators and static analyses of logic programs.

One formal issue that is in need of more research is the theory of recursive categorical equations analogous to the theory of recursive domain equations; the latter has been settled definitively by [Smythe 83].

9. Acknowledgements

We would like to thank Uday Reddy and Esther Shilcrat for helpful discussions, Robert Keller for encouragement and support, and Glynn Winskel for sending us Gordon Plotkin's invaluable unpublished notes on powerdomains. This research has been supported by a University of Utah Graduate Fellowship to Prateek Mishra and by funds provided by IBM Corporation. We acknowledge the support provided to the Dept. Of Computer Science at the University of Utah by the NSF through CER grant # MCS-8121750, which made possible the production of this manuscript.

References

[Abramsky 83] Abramsky, S.
Semantic Foundations of Applicative Multiprogramming.
In Diaz, J. (editor), *Automata, Languages and Programming*, pages 1-14. Springer-Verlag, July, 1983.

[Apt 81] Apt, K., and Plotkin, G.
A Cook's Tour of Countable Nondeterminism.
In *Eighth ICALP*. Springer-Verlag, 1981.

[Cohn 65] Cohn, P.M.
Universal Algebra.
Harper and Row, 1965.

[Cousot 77] P. Cousot and R. Cousot.
Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.
POPL IV :238-252, Jan, 1977.

[Lehmann 76] Lehmann, D.
Categories for Fixed Point Semantics.
Warwick University Theory of Computation Report, Univ. of Warwick, 1976.

[Mishra 84] P. Mishra, R. M. Keller.
Static inference of properties of applicative programs.
In *POPL XI, Salt Lake City*. January, 1984.

[Mycroft 80] A. Mycroft.
The theory and practice of transforming call-by-need into call-by-value.
LNCS 83, Springer-Verlag, 1980, pages 269-281.

[Mycroft 81] A. Mycroft.
Abstract Interpretation and Optimising Transformations for Applicative Programs.
PhD thesis, University of Edinburgh, December, 1981.

[Mycroft 83] A. Mycroft, R. A. O'Keefe.
A polymorphic type system for Prolog.
In *Logic Programming workshop*, pages ???. 1983.

[Panangaden 84] Panangaden, P., and Mishra, P.
Abstraction and Indeterminacy.
Technical Report UUCS-84-006, University of Utah, May, 1984.

[Plotkin 76] G. D. Plotkin.
A Powerdomain Construction.
SIAM J. Comput. 5(3):452-480, Sept., 1976.

[Smythe 78] M. B. Smythe.
Power Domains.
Journal Of Computation And Systems Sciences 16:23–36, 1978.

[Smythe 83] Smythe, M.B.
The Largest Cartesian Closed Category of Domains.
Technical Report, Edinburgh, 1983.

Table of Contents